CS 111 - Homework 8

Deadline

11:59 pm on **Friday, November 4, 2016**

How to submit

Each time you would like to submit your work:

- If your files are not already on nrs-labs, be sure to use sftp/WinSCP/FileZilla to get them TO nrs-labs (ideally in a folder/directory named `111hw8`).
- Then, if you are not already logged onto nrs-labs, then use PuTTY/`ssh` to do so, and use `cd` to change to the folder/directory where your homework files are -- for example,

  ```
  cd 111hw8
  ```

- Use the `ls` command to make sure your desired `.cpp` and `.h` files are really there:

  ```
  ls
  ```

- Use `~st10/111submit` to submit them, with a homework number of **8**
  - Make sure that `~st10/111submit` shows that it submitted **ALL** of your `.cpp` and `.h` files you were intending to submit!

Purpose

To use the design recipe to design and implement C++ functions, including `main` functions, to practice creating "complete" C++ programs, and to write `cout` statements to print to the screen.

Important notes

- As always, start the homework problems early! Then you'll have time to e-mail me your `.cpp` and `.h` files along with the error message(s) if you run into an error you don't know how to handle.
- Be sure to follow the course coding style discussed in class and demonstrated in posted examples. In particular, remember to indent as shown and demonstrated in posted examples.
- You are *no longer required* to use `funct_play` to develop your C++ functions. You *may* still use it if you would like (keeping in mind its limitations for `main` and `void` functions!).
  - If you are not using `funct_play`, note that **you need to include all of the components shown in the given templates** (for `main` functions, non-`main` functions, and `.h` files) on the public course web page, whether you actually use those templates or not.
- Remember that you can JUST compile any single C++ function on nrs-labs with the command:

  `g++ -c` *filename*`.cpp`

  (If all goes well, this doesn't print anything to the screen, but it does result in a file *filename*`.o`, ready to be linked and loaded with other files to create a C++ executable file.)
- Remember that you can compile, link, and load to create a C++ executable program on nrs-labs with the command:

  `g++` *main_filename*`.cpp` *other1*`.cpp` *other2*`.cpp ... -o` *main_filename*

...being sure to include the `.cpp` files for ALL functions used in that program;

- Also, to help you write these `g++` calls to compile, link, and load your C++ programs (to result in C++ executable programs), there is a little script `compile-helper` on nrs-labs that asks you to enter details about your program-to-be and then "builds" an appropriate `g++` command, which it runs as well as prints out so you can gradually learn how to write these yourself.
- (NOTE: it assumes you have already written all of the `.cpp` and `.h` files for your program, and have transferred them all to your current working directory on nrs-labs.)

- You **are, of course, still expected** to follow the Design Recipe for all **functions** that you design/define, whether they are non-`main` or `main` functions.
  - Remember, you will receive **significant** credit for the signature, purpose, header, and examples/tests portions of your functions.
  - Typically you'll get at least half-credit for a correct signature, purpose, header, and examples/tests, even if your function body is not correct.
  - (and, you'll **lose at least half-credit** if you omit these or do them poorly, even if your function body is correct).

Problem 1

FUN FACT: To print a double quote (") or a backslash (\), you have to precede it with a backslash to "escape" its otherwise-special meaning. That is,

```
cout << "moo\"moo\\moo" << endl;
```

causes the following to be printed to the screen:

```
moo"moo\moo
```

With that said...

Consider our `hello` program from Week 10 Lab -- a `main` function that simply prints:

```
Hello CS 111 World!
```

...to standard output/to the screen.

Simply to practice with `cout` within a very simple `main` function, consider "ASCII art" -- drawing "pictures" with "plain" characters.

For example, consider:

```
XX    XX
 XX  XX
   XX
 XX  XX
XX    XX
```

or (very slightly modified from http://www.asciiworld.com/-Cats-.html)

```
 /\_/\
( o o )
 > ^ <
 meow!
```

or (from http://www.incredibleart.org/links/ascii/Scarecrow_Resources.html)

```
\   /
 \o/
```

```
   |
   |
_/ \_
```

Write a `main` function in a file named `draw_pic.cpp` that simply prints to the screen a "hard-coded" ASCII art picture consisting of at least 4 lines of characters.

("Hard-coded" means it does the same thing every time -- it is not customizable, it can't be modified without modifying the program itself. The `main` function in `hello.cpp` could be said to be printing to the screen the "hard-coded" greeting `Hello CS 111 World!`)

You may choose one of the shown examples, or come up with one of your own, or reproduce one you find on the web (IF you include the URL where you found it in a comment within your code -- credit your sources!).

Submit your resulting `draw_pic.cpp`. (NOTE: These *might* be posted on the course Moodle site for class perusal.)

Problem 2

Problem 2 part a

For more practice with `cout`, use the design recipe to write a function `salutation` that expects a name, has the side effect of printing to the screen a letter-style salutation including that name (and ending with a newline), **and** returns the length of the given name. (Yes, it is possible for a function to return something AND have a side-effect -- consider Week 10 Lecture 2's `play_around` function!)

For example, `salutation("Ann") == 3` and has the side-effect of printing to the screen:

```
Dear Ann,
```

and `salutation("Charlie Brown") == 13` and has the side-effect of printing to the screen:

```
Dear Charlie Brown,
```

It is your choice whether you develop this function using `funct_play` or not -- either way, however, you are required to follow the design recipe, including all of the design recipe steps, and, either way, you are expected to write a `main` function, described below, to formally test your function.

Problem 2 part b

To formally test your function, AND to practice writing another `main` function, design a `main` function in a file named `salutation_test.cpp` that:

- prints a message saying that you are testing function `salutation`
- puts `boolalpha` into the `cout` output stream, so that `bool` values are printed as `true` and `false`
- because `salutation` has a desired side-effect in addition to its return value, we need to be more specific about what should be seen as a result of its test calls -- so,

  for EACH of `salutation`'s examples/tests,

  - it should **first** print a message saying that what follows should be a salutation for <print out the name> followed by `true`,
  - and then **put** that example/test in its own separate `cout` statement, such that the result of that test will be printed on its own line

You can compile your `salutation_test.cpp` either by using the `compile-helper` tool on nrs-labs,

or by directly typing the compile command on nrs-labs:

```
g++ salutation_test.cpp salutation.cpp -o salutation_test
```

Submit your resulting files `salutation.cpp`, `salutation.h`, and `salutation_test.cpp`.

Problem 3

Problem 3 part a

Remember that we defined a function `first` in Week 10 Lecture 1 that returns the first character in a `string`. (Or, you can simply grab the first character in a `string` using its `at` method with argument `0` -- take your pick!)

Note, too, that there's a posted example solution for the function `is_vowel` in the Week 9 Lab Exercise in-progress example solutions on the course Moodle site, under "Selected solutions.".

You need to make copies of the files `first.cpp`, `first.h`, `first_ck_expect.cpp`, `is_vowel.cpp`, `is_vowel.h`, and `is_vowel_ck_expect.cpp` in your Homework 8 folder/directory.

Consider function `is_vowel` again. Can you see that, if you had a parameter named `word` whose type is `string`, then it would indeed be the case that:

```
is_vowel(first(word)) == true
```

...if `word` begins with a vowel?

Let's use this to help design a function `pig_lite` which is meant to be a simplistic variation on pig latin. Function `pig_lite` expects a word containing at least one character, and if it starts with a vowel, it produces a string that is that word with `-ay` added to its end -- otherwise, it produces a string that is that word with `-` and its first letter and `ay` added to its end. That is, these should all be true:

```
pig_lite("orange") == "orange-ay"
pig_lite("moo") == "moo-may"
pig_lite("Harold") == "Harold-Hay"
pig_lite("I don't read directions") == "I don't read directions-ay"
```

BUT note...

IF you would like...

...if it bothers you that we aren't removing the first letter of the given word in our result, then you can use Week 10 Lecture 1's `rest` function so that your version does do so.

...OR, if you would like to make some other variation on this, e-mail me with your proposed version, and I'll reply with whether that can be an approved variation on this.

Problem 3 part b

To formally test your function, AND get more practice writing a `main` function, design a `main` function in a file named `pig_lite_test.cpp` that:

- prints a message saying that you are testing function `pig_lite`, and that trues mean passed
- puts `boolalpha` into the `cout` output stream, so that `bool` values are printed as `true` and `false`

- copy each example/test from your function's examples into its own separate `cout` statement, such that the result of that test will be printed on its own line

You can compile your `pig_lite_test.cpp` either by using the `compile-helper` tool on nrs-labs, or by directly typing the compile command on nrs-labs:

```
g++ pig_lite_test.cpp pig_lite.cpp first.cpp is_vowel.cpp -o pig_lite_test
```

Submit your resulting `pig_lite.cpp`, `pig_lite.h`, and `pig_lite_test.cpp`. files.

Problem 4 - (with an additional 10 points BONUS possible)

Problem 4 part a

Consider the example recursive functions from Week 10 Lecture 1, considering a `string` as a list of `char`, with the help of the `first`, `rest`, and `is_empty` functions also created during that lecture.

For some recursion practice, copy the `.cpp` and `.h` files for each of `first`, `rest`, and `is_empty` to your current directory. Then select either ONE (for full credit) or BOTH (for up to 10 points BONUS credit) of the following options (your choice!):

OPTION 1: (easier option) create a recursive function `count_blanks` that uses recursion and these functions -- it should expect a string, and return the number of blank characters in that string. (You only need to count blank characters -- you don't have to worry about any other kind of so-called "white space".)

- for example, `count_blanks("Hi, how are you?") == 3`

  `count_blanks(" oh  my ") == 4`

  `count_blanks("mooo") == 0`

  `count_blanks("") == 0`

OPTION 2: (slightly-more-challenging option) create a recursive function `filter_blanks` that uses recursion and these functions -- it should expect a string, and return a version of that string that has any blanks within it removed/filtered out. (You only need to remove blank characters -- you don't have to worry about removing any other kind of so-called "white space".)

- for example, `filter_blanks("Hi, how are you?") == "Hi,howareyou?"`

  `filter_blanks(" oh  my ") == "ohmy"`

  `filter_blanks("mooo") == "mooo"`

  `filter_blanks("") == ""`

Problem 4 part b

FOR EITHER OPTION: To formally test your function, AND get more practice writing a `main` function, design a `main` function in a file named `count_blanks_test.cpp` or `filter_blanks_test.cpp` (as appropriate) that:

- prints to the screen a message saying that you are about to test that function (give its name!), and that trues mean passed
- put `boolalpha` into the `cout` output stream, so that `bool` values are printed as `true` and `false`
- copy each example/test from your function's examples into its own separate `cout` statement, such that the result of that test will be printed on its own line

You can compile your result either by using the `compile-helper` tool on nrs-labs, or by directly typing the compile command on nrs-labs (each should be typed as a single line, with no carriage return/enter):

```
g++ count_blanks_test.cpp count_blanks.cpp first.cpp rest.cpp
is_empty.cpp -o count_blanks_test
```

or

```
g++ filter_blanks_test.cpp filter_blanks.cpp first.cpp rest.cpp
is_empty.cpp -o filter_blanks_test
```

Submit your resulting `.cpp` and `.h` files.